Target Weak 1:

1. Android Basics: User Interface
2. Android Basics: User Input
3. Android Basics: Multi-screen Apps
4. Android Basics: Networking
5. Android Basics: Data Storage

Day 3/7, Things to cover

- Android Basics: User Interface

This is a basic User Interface Course where I will make “Happy Birthday Card”. It’s quite similar to creating a static website or a wallpaper.

Chapter 1: Building Layout: Part 1

1.1 Welcome to the course

Prerequisite: None
Target: 2 Apps

1.2 Introduction

Target of the course:

1. Make a single screen Android app with texts and images.
2. Learning how to use Android Studio
3. Always keeping in mind that User Interface must target to the audience in the right way.

1.3 Preparing for the Journey

1.4 Views

Layout of an app is a blueprint of the app. It is the design or arrangement of what user sees.

A **View** is the basic building block that we use to build the layout of an app. It is a rectangular area on the screen and has a width and a height.

A View can be of many types

1. TextView
2. ImageView - Used to display the image.
3. Button - Used to display button with labelled text.
4. View - Plain rectangle which can be used as a divider.

5. EditText - TextView that we can type into.
6. Spinner - Dropdown Option menu.
7. Checkbox - Where you can select more than one options.
8. RadioButton - Where you can select one of the given options.
9. RatingBar - Used to put star rating
10. Switch -On/Off switch that we can drag left or right
11. SeekBar - Displays progress and allows us to drag the handle anywhere
12. SearchView -A search field where we can type the query into
13. ProgressBar - Loading spinner

I also learnt about CamelCase convention.

1.5 YouTube App

Discussion about various Views in the Youtube App.

1.6 Talking to your friend

IDE: Integrated Development Environment
XML: eXtensible Markup Language
Android Studio

1.7 Using TextView

I learnt about various attributes of the TextView like **android:layout_width, android:layout_height, android:text and android:background**.

1.8 XML Syntax

1. XMl elements
2. Tag
3. Self Closing tag
4. Attributes
5. Syntax

```
<TextView
      android:layout_width=""
      android:layout_height=""
      android:background=""
      android:text=""
      android:textStyle=""
      />

<LinearLayout
      android:layout_width=""
      android:layout_height="">

      <TextView
            android:layout_width=""
            android:layout_height=""
```

```
            android:background=""
            android:text=""
            />

      <TextView
            android:layout_width=""
            android:layout_height=""
            android:background=""
            android:text=""
            />
</LinearLayout>
```

1.9 Change TextView

Here I learnt about two different kind of scales of measurement

1. **DP (Density Independent Pixel)**
2. **SP (Scale Independent Pixel)**

The screen of an android is made up of rows and columns of glowing dots called **Pixel**. Devices can range in screen density which means how many pixel per inch are on the screen. For ex. Medium Density Device (mdpi) have 160 pixels/inch but for Extra Extra High Density Devices (xxhdpi) have 480 dots/inch

If we specify the size of view in pixels it would appear small on high density devices where and large on small density devices which would create inconsistency. If a button is too small it will be difficult for a user to touch.

To achieve the consistency in the physical sizes of the views across all the devices we use another scale of measurement called Density Independent Pixels (DP).

A DP is a mapper. 1 DP = 1px on mdpd = 3px on xxhdpi and so on. Android devices will automatically handle the conversion and we don't need to take care of it.

1.10 Getting past errors

1.11 Setting wrap_content

There are three ways to change the size of the View

1. Hard Coding where we have to specify the actual size
2. **wrap_content**
3. **match_parent**

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

1.12 TextView textSize

A scale-independent pixel (sp) is a unit of length for specifying the size of a font of type. Its length depends on the user's preference for font size, set in the Settings app of the Android device.

To respect the user's preferences, you should specify all font sizes in scale-independent pixels. All other measurements should be given in device-independent pixels (dp's).

1.13 TextView textColor

```
android:textColor="#____"
android:textColor="@android:color/____"
```

1.14 Simple ImageView

```
<ImageView
      android:src="@drawable/___"
      android:layout_height=""
      android:layout_width=""
      android:scaleType=""
      />
```

1.15 Documentation

developer.android.com

Chapter 2: Building Layout: Part 2

2.1 ViewGroups

A ViewGroup is a big View that can contain smaller Views inside of it.
The biggest of all the views which encloses all the other views is called **Root view**. The smaller Views are called the **children** of the ViewGroup. . The ViewGroup is called the **parent** of its children.

2.2 Types of ViewGroups

There are three kind of layouts which we use

1. LinearLayout - is a commonly used ViewGroup and let's us arrange its children view in vertical column or horizontal row.
2. RelativeLayout - lets us position its children views relative to its own edges.
3. ConstraintLayout*

2.3 LinearLayout

```
<LinearLayout
android:layout_width=""
android:layout_height=""
android:orientation="" >
<TextView
android:layout_width=""
android:layout_height=""
android:background=""
android:text=""
/>
      <TextView
            android:layout_width=""
            android:layout_height=""
            android:background=""
            android:text=""
            />
</LinearLayout>
```

By default the orientation of LinearLayout is horizontal but we can use vertical orientation as well.

2.4 Width and Height

There are three ways to change the size of the View

1. Hard Coding (Not recommended)
2. wrap_content
3. match_parent

2.5 Evenly Spacing out of Children

LinearLayout is a ViewGroup that aligns all the children in a single direction, vertical or horizontal. We can specify the alignment by using `android:orientation` attribute of LinearLayout.
All children are stacked one after the other. So a vertical stack will only have one children per row and a horizontal stack will have one children per column.
By default all the children are equally weighted (0 weight). It means that all of the children will be given equal priority.

2.6 Inner Layout Weight

Every view in the LinearLayout can request for additional width for itself. First we allocate necessary space to all the views. Once all the views are filled in position, if any space left it will be used to fulfill the additional space requests of views.

```
<LinearLayout
      android:layout_width=""
      android:layout_height=""
      android:orientation="" >
      <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:src="@drawable/ic_chat"
            />
      <EditText
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:hint="Send a message"
            />

      <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:src="@drawable/ic_send"
            />
</LinearLayout>
```

Above code is for Horizontal LinearLayout

2.7 RelativeLayout

RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID).
Important Attributes

A. android:layout_alignParentLeft
B. android:layout_alignParentRight
C. android:layout_alignParentTop
D. android:layout_alignParentBottom
E. android:layout_centerHorizontal
F. android:layout_centerVertical
G. android:layout_centerInParent
H. android:layout_toLeftOf
I. android:layout_toRightOf
J. android:layout_above
K. android:layout_below

2.8 ID

Any view group may have integer ID associated with it, to uniquely identify the views within the tree. When an app is compiled, the ID is references as an integer, but the ID is typically assigned in a large XML file as a string in the ID attribute

```
android:id="@+id/name_of_the_id"
```

@ - XML parser should parse and expand the rest of the ID string and identify it as an ID resource
+ - This means that a new resource must be created and added to the resource.

2.9 Relative to other view

Positioning children relative to other views.

1. android:layout_toLeftOf
2. android:layout_toRightOf
3. android:layout_above
4. android:layout_below

2.10 List Items with RelativeLayout

2.11 Padding vs Margin

A view will wrap itself around its content if we set the view's width and/or height to special value "wrap_content". To prevent it from wrapping too tightly, we can specify an amount of padding on each side.

```
android:padding
android:paddingLeft
android:paddingRight
android:paddingTop
android:paddingBottom
```

If we don't want boundaries of two views, who are placed adjacent to each other, not to touch then we can ask for margin on sides.

```
android:layout_margin
android:layout_marginLeft
android:layout_marginRight
android:layout_marginTop
android:layout_marginBottom
```

Chapter 3 Practice Set: Building Layout

3.1-3.9 Introduction and setting up environments

1. Java Development KIt (JDK)
2. Android Studio

3.10 Hello World !!

```
public class Hello
 {
     public static void main(String args[ ])
     {
          System.out.println("Hello World !!");
     }

}
```

3.11-3.12 Getting started with Android Studio

1. Start a new Application
2. Application Name, Company Domain, Project Location
3. Phone & Tablet - API 15: Android 4.08 (Ice cream Sandwich)
4. Empty Activity
5. Starting Template - Blank Activity
6. Customize Activity

3.13 Running Hello World on your Phone

1. Become a Developer by turning on the developer mode on the Phone.
2. Connect your Phone to your Computer
3. Install Drivers (Windows only)
4. Install App

3.14 Using Emulator

1. Phone as Emulator
2. Virtual Device as Emulator (*) - Since I have problem with my Android Phone, I will be using virtual Emulator only.

3.15 Creating a Birthday card app
Step 1: Figure out the Views you'll be using.
Step 2: Position Views as needed.
Step 3 Style Views.

3.16 LinearLayout and RelativeLayout

3.17 The Drawable Folder

3.18 Positioning

3.19 Styling

3.20 Making the text larger

```
android:textSize="__sp"
```

3.21 Setting the Font

```
android:fontFamily="sans-serif-light"
```

3.22 Setting the Color

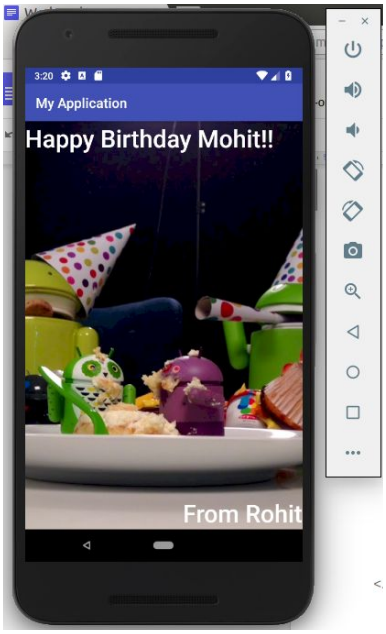
```
android:textColor="#________"
```

3.23 -3.24 attributes of ImageView | Styling the Image

1. `android:maxHeight` - An optional argument to supply a maximum height for this view.
2. `android:maxWidth` - An optional argument to supply a maximum width for this view.
3. `android:scaleType` - Controls how the image should be resized or moved to match the size of this ImageView.
4. `Android:src` - Sets a drawable as the content of this ImageView.

3.25-3.31 Final Touch

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <ImageView
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:src="@drawable/androidparty"
       android:scaleType="centerCrop"/>
   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Happy Birthday Mohit!!"
       android:textStyle="bold"
       android:textSize="36sp"
       android:textColor="@android:color/white"
       android:fontFamily="sans-serif-light"
       android:layout_alignParentTop="true"
       android:layout_alignParentLeft="true"/>
   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="From Rohit"
       android:textStyle="bold"
       android:textSize="36sp"
       android:textColor="@android:color/white"
       android:fontFamily="sans-serif-light"
       android:layout_alignParentBottom="true"
       android:layout_alignParentRight="true"/>
</RelativeLayout>
```


3:20
My Application
Happy Birthday Mohit!!
From Rohit